System wtyczek w oparciu o interfejsy.

Reichel Bartosz
email:reichel@reichel.pl
www.reichel.pl

2006.08.19

1 Wstęp

System wtyczek (potocznie zwanych pluginami) można zaprogramować na wiele sposobów. Począwszy od haków na okienka poprzez biblioteki z zestawem funkcji czy też eksportujące tablice z zestawem funkcji skończywszy na rozwiązaniu, moim zdaniem najbardziej uniwersalnym, a mianowicie na interfejsach. Wtyczka oparta o interfejs łączy zalety wszystkich pozostałych opcji a w istocie jest to tylko wygodne przedstawienie danych w trakcie tworzenia aplikacji gdyż po skompilowaniu zaimplementowany interfejs można utożsamiać z rekordem zawierającym wskazniki do poszczególnych funkcji (tablice funkcji - VMTable, virtual method table).

Przedstawiony tutaj przykład jest podpatrzony z systemu windows i jego rozszerzeń. Wydaje mi się on najbardziej uniwersalny pod względem możliwości implementacji oraz jej łatwości w innych językach. Naturalnie można poszerzyć to rozwiązanie o biblioteki typów (TypeLib) czy też o komponenty w postaci kontrolek ActiveX (jednak w tym przypadku twórca wtyczki ma bardzo dużą kontrolę nad aplikacją i ciężko ją ograniczyć).

2 Co nieco o COM

Jak już wspomniałem idea wtyczek przedstawiona tutaj, jest oparta na wtyczkach systemu windows. Zatem stosować tu będę podobną konwencje. Każda z funkcji w interfejsie zwraca rezultat (**HResult**), który może przyjąć wartość **S_OK** w przypadku powodzenia oraz inne wartości w przypadku błędu (np. **E_FAIL**, **E_NOTIMPL**, ...) więcej w **Windows.pas**

Dzięki takiemu podejściu możemy wykorzystać funkcje już zdefiniowane w większości języków do sprawdzanie czy funkcja zwróciła wartość poprawną. W delphi są to funkcje **SUCCEEDED**, **OleCheck**, **FAILED**.

Listing 1: fragment pliku ActiveX.pas

```
{$EXTERNALSYM Succeeded}
function Succeeded(Res: HResult): Boolean;
{$EXTERNALSYM Failed}
function Failed(Res: HResult): Boolean;
{$EXTERNALSYM ResultCode}
function ResultCode(Res: HResult): Integer;
{$EXTERNALSYM ResultFacility}
function ResultFacility(Res: HResult): Integer;
{$EXTERNALSYM ResultSeverity}
```

```
function ResultSeverity(Res: HResult): Integer;
{$EXTERNALSYM MakeResult}
function MakeResult(Severity, Facility, Code: Integer): HResult;
```

Nie będę tutaj opisywał szczegółów związanych z posługiwaniem się interfejsami odsyłam do książek [1] [2].

Jedyne na co chciałbym zwrócić uwagę to na niszczenie interfejsow. Rozpatrzmy interfejs IFoo oraz jego wersje zapakowaną razem z obiektem (TInterfacedObject) TFoo.

```
type
  IFoo = interface(IInterface)
  [SID_IFoo]
end;
type
  TFoo = class(TInterfacedObject, IFoo);
end;
```

teraz dwa przypadki niszczenia, pierwsza z deklaracja zmiennej Foo jako interfejsu:

```
var
 Foo:IFoo;
begin
   Foo := TFoo.Create;
   //tu pracujemy z interfejsem
end;//na koniec funkcji zostanie on zniszczony
```

teraz drugi przypadek, w którym zmieniamy tylko deklaracje zmiennej Foo:

```
var
 Foo:TFoo;
begin
   Foo := TFoo.Create;
   //tu pracujemy z interfejsem
end;//na koniec funkcji !!!! nie !!!! zostanie on zniszczony
```

3 Interfejs - wtyczka

Projektując interfejs dla wtyczek powinniśmy się zastanowić jak dużą władze nad naszą aplikacją chcemy dać programistom (Lepiej jej dać za mało, bo łatwiej jest dodać uprawnienia niż je zabrać - kompatybilność).

Następnie musimy stworzyć opis takiego interfejsu jak np. przedstawiony tutaj (troche sztuczny).

Listing 2: definicja interfejsu wtyczki i aplikacji rodzica

```
//************************************
//          Reichel Bartosz
//          reichel@mif.pg.gda.pl
//          http://reichel.pl
//          2006.08.19
//************************************
unit MyInterface;
```

```
interface
uses Windows, ActiveX, COmObj;

//Scieżka w rejestrze, pod którą należy rejestrować
//nowe rozszerzenia. Należy utworzyć klucz z numerem
//GUID np:
//SOFTWARE\PFS\Examples\Wtyczki\{000C0AF6-5B6E-47BF-90F2-82E15480AB96}
//
const
 //HKEY_CURRENT_USER
 szWtyczki   = 'SOFTWARE\PFS\Examples\Wtyczki';


const
 SID_IPlugsApplication = '{822C0AF6-5B6E-47BF-90F2-82E15480AB96}';
 IID_IPlugsApplication: TGUID = '{822C0AF6-5B6E-47BF-90F2-82E15480AB96}';

//*****************  IPlugsApplication *****************
//------------
//function ChangeColor(NewColor:TColorRef):HResult;stdcall;
//
//funkca ChangeColor pozwala zmienić,
//kolor tła aplikacji rodzica. Jako parametr kolor w postaci RGB.
//------------
//function GetHandle(out hwnd:HWND):HResult;stdcall;
//
//funkcja GetHandle zwraca uchwyt okna rodzica w parametrze hwnd
//*****************************************************
type
  IPlugsApplication = interface(IInterface)
  [SID_IPlugsApplication]
  function ChangeColor(NewColor:TColorRef):HResult;stdcall;
  function GetHandle(out hwnd:HWND):HResult;stdcall;
end;

const
 SID_IPlug = '{E2F26AD7-4A48-4CE4-8CC8-76192E0E9640}';
 IID_IPlug: TGUID = '{E2F26AD7-4A48-4CE4-8CC8-76192E0E9640}';

 SID_IPlugSuper = '{E9F1E678-C99B-46C2-A51F-AA4197182C73}';
 IID_IPlugSuper: TGUID = '{E9F1E678-C99B-46C2-A51F-AA4197182C73}';

//********************** IPlug **********************
//*****************************************************
//function Init(ap:IPlugsApplication;TabH:HWND): HResult; stdcall;
//
//W przypadku gdy implementujemy interfejs IPlugsApplication
//
//Funkcja Init jest najważniejszą częścią składową
```

```
//wtyczki opartej o IPlug i musi być zaimplementowana.
//Jako parametr należy podać istniejący wskaźnik do
//interfejsu aplikacji rodzica IPlugsApplication. W przeciwny
//wypadku funkcja zwróci wartość E_FAIL.
//
//Dla osób implementujących interfejs IPlug.
//
//Funkcja Init musi być zaimplementowana.
//W parametrze ap powinien znajdować się wskaźnik do
//interfejsu IPlugsApplication. Jeśli przyjmuje on
//wartość pustą funkcja Init powinna zwrócić kod
//E_FAIL i zakończyć inicjacje. W przypadku gdy wskaźnik
//jest poprawny, to należy w tej funkcji utworzyć okno
//wtyczki. W przypadku gdy parametr TabH jest różny od
//zera, okno może być utworzone z parametrem WS_CHILD.
//Jako wartość określającą powodzenie należy zwrócić S_OK
//----------------
//
// function Mouse(Name:PChar;X,Y:Integer): HResult; stdcall;
//
//Dla implementujących IPlugsApplication
//
//Do funkcji można przekazać parametry:
//       Name - opisujący nad czym znajduje się kursor
//       X,Y  - pozycje kursora myszy
//
//Dla implementujących interfejs IPlug
//
//Funkcja może być nie zaimplementowana i zwrócić E_NOIMPL.
//W przypadku gdy funkcja jest zaimplementowana
//można reagować na parametry Name, X,Y w zależności
//od zastosowania wtyczki.
//
//
//function GetName(var Desc:PChar):HResult; stdcall;
//
//Funkcja przekazuje dodatkowe informacje o wtyczzce.
//
//Funkcja GetName może być nie zaimplementowana i zwrócić
//wartość E_NOIMPL. W przypadku gdy jest zaimplementowana
//Powinna zarezerwować odpowiednią ilość pamięci dla zmiennej
//Name. Za zwolnienie pamięci odpowiedzialny jest wywołujący
//funkcje.
//
//
//function FreePlug(): HResult; stdcall;
//
//Funkcja FreePlug zwalnia zasoby pamięci zarezerwowane
//podczas działania wtyczki.
```

```
//
//Powinna zwrócić wartość S_OK.
//
//******************************************************
//******************************************************
type
  IPlug = interface(IInterface)
    [SID_IPlug]
    function Init(ap:IPlugsApplication;TabH:HWND): HResult; stdcall;
    function Mouse(Name:PChar;X,Y:Integer): HResult; stdcall;
    function GetName(var Desc:PChar):HResult; stdcall;
    function FreePlug(): HResult; stdcall;
  end;

//********************** IPlugSuper ********************
//******************************************************
//Interfejs IPlugSuper jest rozszerzeniem interfejsu
//IPlug. Implementujący ten interfejs powinien
//też zaimplementować funkcje interfejsu IPlug.
//
//
//function Super(): HResult; stdcall;
//
//Funkcja Super powinna zwrócić S_OK i zrobić coś Super!
//
//******************************************************
//******************************************************

type
  IPlugSuper = interface(IPlug)
    [SID_IPlugSuper]
    function Super(): HResult; stdcall;
  end;
implementation


end.
```

Dobrym zwyczajem jest podanie jakiegoś bardzo prostego przykładu wtyczki (coś w rodzaju SDK - software development kit, dla naszej wtyczki).

Należy szczególnie opisać jak należy reagować w przypadku błędu. Gdyż później nieprawidłowo napisane wtyczki mogą przyczynić się do niestabilności całej aplikacji.

4 Aplikacja - rodzic

Stworzenie dobrej aplikacji rodzica jest chyba najtrudniejszą sprawę. Programista piszący taką aplikację musi przewidzieć posunięcia przyszłych twórców wtyczek. Część związana z wywołaniem funkcji wtyczek powinna cechować się dobrą kontrolą błędów. Programista powinien przewidzieć nawet absurdalne posunięcia twórcy wtyczki i odpowiednio zareagować (np. usunąć wtyczkę z pamięci).

Chcielibyśmy jeszcze umożliwić komunikację wtyczki z aplikacją w obie strony. Dobrym do tego celu obiektem jest również interfejs, w tym przypadku przedstawiony jako **IPlugsApplication**. Pozwala on kontrolować kolor aplikacji oraz pobierać uchwyt okna głównego. Natomiast w drugą strone aplikacja rodzic wywołuje funkcje interfejsu **IPlug** lub **IPlugSuper**.

Definicja naszego formularza aplikacji rodzica powinna zatem zawierać w sobie deklaracje interfejsu, przedstawia to poniższy listing:

```
type
  TForm1 = class(TForm, IPlugsApplication)
    ListView1: TListView;
    OpenDialog1: TOpenDialog;

    .....

//IPlugsApplication
    function ChangeColor(NewColor:TColorRef):HResult;stdcall;
    function GetHandle(out hwnd:HWND):HResult;stdcall;
  public
    { Public declarations }
  end;
```

Natomiast funkcje (tu zmieniajaca kolor formularza i pobierająca uchwyt) powinny wyglądać w następujący sposób

```
function TForm1.ChangeColor(NewColor:TColorRef):HResult; begin
  Color := NewColor;
  result := S_OK;
end;

function TForm1.GetHandle(out hwnd:HWND):HResult; begin
 // hwnd := panel1.Handle;
  result := S_OK;
end;
```

Zgodnie ze wcześniejszym opisem interfejsu obie powinny zwrócić **S_OK**.

Przejdźmy teraz do najważniejszej funkcji **ListPlugs**, wyświetla ona liste dostępnych wtyczek. Lista ta znajduje się w rejestrze pod ścieżką *SOFTWARE*
PFS
Examples
Wtyczki. Każda wtyczka reprezentowana jest jako numer CLSID (ten unikalny numer określający komponent COM, można wygenerować w Delphi poprzez wciśnięcie kombinacji klawiszy Ctrl+Shift+G).

Najbardziej istotną funkcją jest tu funkcja **CoCreateInstance**. Ładuje ona naszą wtyczkę do pamięci (w przypadku tutaj przedstawionym uruchamiana jest również funkcja **Initialize** w implementacji wtyczki). Jako parametry tej funkcji należy podać numer GUID (ang. Globally Unique Identifier). Po więcej szczegółów o funkcji COCreateInstance odsyłam do MSDN

Listing 3: Wywołanie funkcji CoCreateInstance

```
HR := CoCreateInstance(WtykaGUID, nil ,CLSCTX\_INPROC\_SERVER, IID\_IPlug ,W);
```

Jako interfejs za pomocą, którego będziemy się komunikowali z naszą wtyczką wybieramy GUID określający interfejs IPlug czyli IID_IPlug. Ten interfejs jest najbardziej podstawowy dla naszej wtyczki i każda wtyczka powinna posiadać jego implementacje.

Jeśli teraz chcemy przekonać się czy nasza wtyczka zawiera jakieś dodatkowe interfejsy, możemy wywołać funkcje **QueryInterface** z instancji utworzonej wtyczki. W naszym przypadku możemy oczekiwać, że wtyczka zawiera rozszerzenie w postaci interfejsu IPlugSuper. Zatem wywołujemy funkcje QueryInterface z parametrem IID_IPlugSuper. Jeśli funkcja zwróci rezultat S_OK, oznaczać to będzie, że wtyczka zawiera dodatkowy interfejs (a jego instancja znajduje się w zmiennej SW:IPlugSuper).

W funkcji ListPlugs, kod niszczący obiekty jest wywoływany automatycznie.

Listing 4: Funkcja ListPlugs

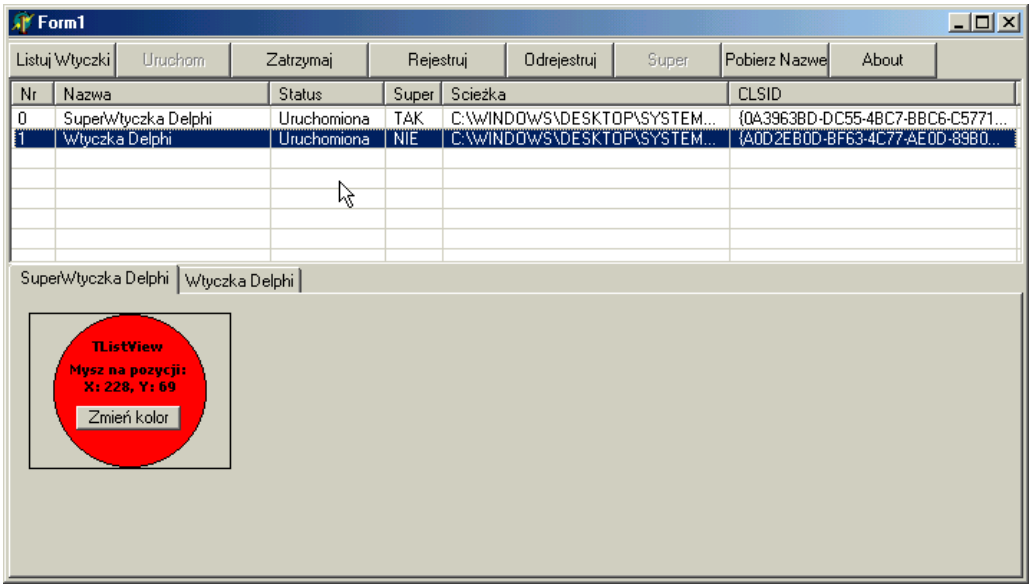
```
procedure TForm1.ListPlugs;
var
 Reg:TRegistry;
 SL:TStringList;
 i:integer;
 WtykaGUID:TGUID;
 HR:HResult;
 PR:PPlugs;
 W:IPlug;
 Wname,Wpath:String;
 LI:TListItem;
 SW:IPlugSuper;
begin
    Screen.Cursor := crHourGlass;
    Button2.Enabled := False;
    ListView1.Items.BeginUpdate;
    try
     Reg := TRegistry.Create;
     try
      Reg.RootKey := HKEY_CURRENT_USER;
      Reg.OpenKey(szWtyczki,True);
      SL := TStringList.Create;
      try
       Reg.GetKeyNames(SL);
       if SL.Count > 0 then
        for i := 0 to SL.Count - 1 do
        begin

         if not CheckForPlug(SL[i]) then
         begin
         WtykaGUID := StringToGuid(SL[i]);
         HR := CoCreateInstance(WtykaGUID,nil,CLSCTX_INPROC_SERVER,IID_IPlug,W);
         if SUCCEEDED(HR) then
         begin
          GetInfoFromCLSID(WtykaGUID,Wname, Wpath);
          LI := ListView1.Items.Add;
          New(PR);
          PR^.Int := nil;//na razie interfejsu nie ma - jest niezaladowana
          PR^.Loaded := False;
          PR^.IsSuper := False;
          PR^.Path := WPath;
```

```
            PR^.Name := WName;
            PR^.CLSID := SL[i];
            LI.Data := PR;
            LI.Caption := IntToStr(i);
            LI.SubItems.Add(Wname); //nazwa
            LI.SubItems.Add(cStatus[False]);
            if SUCCEEDED(W.QueryInterface(IID_IPlugSuper,SW)) then
            begin
             LI.SubItems.Add('TAK'); //Czy super wtyczka
             PR^.IsSuper := True;
            end
            else
             LI.SubItems.Add('NIE');

            LI.SubItems.Add(Wpath); //scieżka
            LI.SubItems.Add(SL[i]);    //CLSID
           W := nil;
          end;
          end;
        end;
       finally
        SL.Free;
       end;
      finally
       Reg.Free;
      end;
     finally
      ListView1.Items.EndUpdate;
      Button2.Enabled := True;
      Screen.Cursor := crDefault;
     end;
end;
```

Wciskając guzik uruchom, również wywołujemy funkcje CoCreateInstance, jednak tym razem wskaźnik do interfejsu przechowywujemy w rekordzie PPlugs. Wskaźnik do tego rekordu przechowywujem w liście ListView1. W procedurze LoadPlugClick, poza tym, że tworzymy instancje interfejsu to jeszcze wywołujemy funkcje **Init**. Jako parametry tej funkcji podajemy interfejs IPlugsApplication wskazujący na nasz formularz oraz uchwyt okna, na którym ma być utworzone okno naszej wtyczki (w przedstawionym przykładzie jest to uchwyt okna zakładki TTabSheet).

Listing 5: Funkcja uruchamiająca wtyczkę.

```
procedure TForm1.LoadPlugClick(Sender: TObject);
var
 LI:TListItem;
 PR:PPlugs;
 PlugGUID:TGUID;
 HR:HResult;

begin
```



Rysunek 1: Aplikacja rodzic w akcji

```
if not Assigned(ListView1.Selected) then exit;
LI := ListView1.Selected;
PR := PPlugs(LI.Data);
PlugGUID := StringToGuid(PR^.CLSID);

HR := CoCreateInstance(PlugGUID,nil,CLSCTX_INPROC_SERVER,IID_IPlug,PR^.Int);
if SUCCEEDED(HR) then
begin
//Tworzenie nowej zakładki
      PR^.Tab := TTabSheet.Create(PageControl1);
      PR^.Tab.PageControl := PageControl1;
      PR^.Tab.Caption := PR^.Name;
      If SUCCEEDED(PR^.Int.Init(Form1 as IPlugsApplication, PR^.Tab.Handle)) then
        PR^.Loaded := true
      else
        PR^.Tab.Free;//W razie niepowodzenia należy ją usunąć.
end;

ListView1.SetFocus;
ListView1.Selected := LI;
ListView1.ItemFocused := LI;
ListView1Click(Sender);
end;
```

W związku z tym, że wskaźniki do interfejsów są przechowywane w liście ListView1 jesteśmy zobowiązani zwolnić je sami. Niestety podczas destrukcji komponentów Delphi, nie zwalnia pamięci przydzielonej do pola Data poszczególnych pozycji w liście (TListItem).

Należy tu zwrócić uwagę na sprawdzanie warunków logicznych. Powinna być wyłączona dysrek-

tywa **B** kompilatora (Boolean short-circuit evaluation). Gdyż sprawdzenie *PR.Loaded* w przypadku gdy PR = nil, może spowodować błąd.

Listing 6: Destruktor aplikacji rodzica

```
procedure TForm1.FormDestroy(Sender: TObject);
var
 I:Integer;
 PR:PPlugs;
begin
//W przypadku gdy interfejsy sa gdzieś 'zamotane' w pamięci
//nalezy je zwolnić samemu !!


if ListView1.Items.Count > 0 then
 for i:= 0 to ListView1.Items.Count -1 do
 begin
  PR := ListView1.Items[i].Data;
 //uwaga gdy włączone sprawdzanie wszystkich warunków to się posypię >> {$B-}
  if Assigned(PR) and PR^.Loaded then
  begin
    PR^.Int := nil; //zwalniamy wtyczke (to samo co _Release - nie pamietam, od ktorej
    PR^.Tab.Free;
  end;
  if Assigned(PR) then Dispose(PR);//no i zwalniamy pamięć przydzieloną do rekordow
 end;

   pMalloc := nil;
   CoUnInitialize;
end;
```

Wywołanie metody z interfejsu IPlugSuper dziedziczącego z IPlug, można rozwiązać za pomocą rzutowania lub za pomocą słowa kluczowego Delphi **as**

Listing 7: Funkcja wywołująca funkcje Super.

```
procedure TForm1.SuperClick(Sender: TObject);
var
 LI:TListItem;
 PR:PPlugs;
begin

 if not Assigned(ListView1.Selected) then exit;
 LI :=ListView1.Selected;
 PR := PPlugs(LI.Data);
 if not Assigned(PR) then exit;

 (PR^.Int as IPlugSuper).Super();

end;
```

Funkcja pobierająca nazwę jest podana jako przykład wywołania wyjątku za pomocą funkcji **OleCheck**. Efekt działania przedstawiony na rysunku 2.

Listing 8: Funkcja pobierająca dodatkowe info o wtyczce.

```
procedure TForm1.Button1Click(Sender: TObject);
var
 LI:TListItem;
 PR:PPlugs;
 Name:PChar;
begin
if not Assigned(ListView1.Selected) then exit;
LI:=ListView1.Selected;
PR := PPlugs(LI.Data);
if Assigned(PR) and PR^.Loaded then
begin
 OleCheck(PR^.Int.GetName(Name));
 MessageBox(handle,name,'',MB_OK);
 pMalloc.Free(Name);
end;
end;
```

(a) (b)

Rysunek 2: a) W przypadku gdy funkcja jest zaimplementowana, pokaże się komunikat z tekstem zwróconym przez funkcje GetName b) komunikat dla prostej wtyczki - metoda niezaimplementowana

w przedstawionym tutaj przykładzie w funkcji OnMouseMove, listowane są wszystkie wtyczki i wysyłana jest do nich informacja, poprzez funkcje Mouse(name,X,Y), o położeniu kursora myszy oraz o nazwie komponentu nad jakim znajduje się kursor.

Listing 9: Obsługa ruchu myszy.

```
procedure TForm1.Button1MouseMove(Sender: TObject; Shift: TShiftState; X,
  Y: Integer);
var
 I:Integer;
 PR:PPlugs;
 SC:String;
begin
```

```
if ListView1.Items.Count > 0 then
 for i:= 0 to ListView1.Items.Count -1 do
 begin
  PR := ListView1.Items[i].Data;
  if Assigned(PR) and PR^.Loaded then
  begin
   if Assigned(Sender) then
   begin
    SC := Sender.ClassName;
    if assigned(PR^.Int) then PR^.Int.Mouse(Pchar(SC),X,Y);
   end;
  end
 end;

end;
```

Funkcja RegisterInterfaces rejestruje nazwy interfejsów w rejestrze. W takim przypadku inne programy mając numer CLSID, mogą przedstawić nazwę interfejsu w bardziej czytelnej formie (Patrz program OleView) [3].

Listing 10: Rejestracja interfejsów.

```
procedure RegisterInterfaces;
var
 Reg:TRegistry;
begin
   Reg := TRegistry.Create;
   try
    Reg.RootKey := HKEY_CLASSES_ROOT;
    Reg.OpenKey('Interface\'+SID_IPlugsApplication,True);
    Reg.WriteString('','IPlugsApplication');
    Reg.CloseKey;

    Reg.OpenKey('Interface\'+SID_IPlug,True);
    Reg.WriteString('','IPlug');
    Reg.CloseKey;

    Reg.OpenKey('Interface\'+SID_IPlugSuper,True);
    Reg.WriteString('','IPlugSuper');
    Reg.CloseKey;
   finally
    Reg.free;
   end;
end;
```

5 Tworzenie wtyczki

Aby utworzyć wtyczke w postaci biblioteki dll, trzeba zaimplementować interfejs IPlug. Można tego dokonać w Delphi, dobudowywując go do specjalnie do tego przeznaczonego obiektu TComObject.

Listing 11: Definicja klasy wtyczki.

```
type
  TDelphiWtyczka = class(TComObject, IPlug)
  private
    FWind: TForm1;
  public
//IPlug
    function Init(ap:IPlugsApplication;TabH:HWND): HResult; stdcall;
    function Mouse(Name:PChar;X,Y:Integer): HResult; stdcall;
    function GetName(var Desc:PChar):HResult; stdcall;
    function FreePlug(): HResult; stdcall;
 public
    procedure  Initialize; override;
    destructor Destroy; override;
  end;

const

  CLSID_DelphiWtyczka: TGUID = '{A0D2EB0D-BF63-4C77-AE0D-89B0234711C9}';
```

Obiekt ten posiada dodatkowo jeszcze funkcję wywoływaną podczas inicjacji - **Initialize**. W destruktorze powinniśmy usunąć wszelkie obiekty z pamięci.

```
procedure TDelphiWtyczka.Initialize;
begin
//Tutaj incjujemy elementy potrzebne do działania naszego pluginu
  inherited;
end;

destructor TDelphiWtyczka.Destroy;
begin
  if Fwind <> nil then FWind.Close;
  inherited;
end;
```

W funkcji Init, która musi być zaimplementowana do poprawnego dzoiałania naszej wtyczki, powinniśmy utworzyć okno i umieścić je na oknie rodzicu (uchwyt podany w parametrze TabH). Naturalnie w naszych wtyczkach nie jesteśmy zmuszeni tworzyć okna, możemy np. pobierać uchwyt kontekstu okna rodzica i malować po nim za pomocą dostępnych w danym języku funkcji (lub też bezpośrednio wykorzystywać funkcje WinAPI z modułu GDI).

Listing 12: Funkcja inicjująca okno wtyczki.

```
function TDelphiWtyczka.Init(ap:IPlugsApplication;TabH:HWND): HResult;
begin
//zakładamy, że aplikacja musi mieć interfejs aplikacji.
    if ap = nil then
    begin
      result := E_FAIL;
      Exit;
    end;
```

```
    Fwind := TForm1.CreateP(TabH);
    Fwind.top := 10;
    Fwind.left := 10;
    Fwind.show;
    result := S_OK;
end;
```

Przedstawiony przypadek jest minimalistyczny, więc w funkcji Mouse zwracamy E_NOTIMPL.

```
function TDelphiWtyczka.Mouse(Name:PChar;X,Y:Integer): HResult;
begin
// Jeśli zwrócimy, ze metoda jest niezaimplementowana
//aplikacja rodzic może zareagować na to w odpowiedni sposób.
//W najgorszym przypadku np stosując funkcje OleCheck
//może zwrócić wyjątek, ze metoda jest niezaimplementowna.
 result := E_NOTIMPL;
end;
```

Podobnie czynimy z funkcją GetName. Natomiast w funkcji FreePlug zamykamy okno.

```
function TDelphiWtyczka.GetName(var Desc:Pchar):HResult;
begin
 result := E_NOTIMPL;
end;

function TDelphiWtyczka.FreePlug(): HResult;
begin
  FWind.Close;
  Result := S_OK;
end;
```

Aby naszą bibliotekę można było zarejestrować powinna ona eksportować zestaw funkcji specyficznych dla interfejsów COM.

Listing 13: Niezbędne eksporty biblioteki DLL

```
exports
  DllGetClassObject,
  DllCanUnloadNow,
  DllRegisterServer,
  DllUnregisterServer;
```

Dodatkowo powinna zawierać tzw. Fabrykę Klas (ClassFactory związane z interfejsem IClassFactory). W Delphi (i w większości większych kompilatorów) jest już zaimplementowana klasa ClassFactory. My możemy ją rozbudować o dodatkową rejestracje naszej biblioteki w rejestrze dla potrzeb aplikacji rodzica.

Listing 14: Fabryka klas.

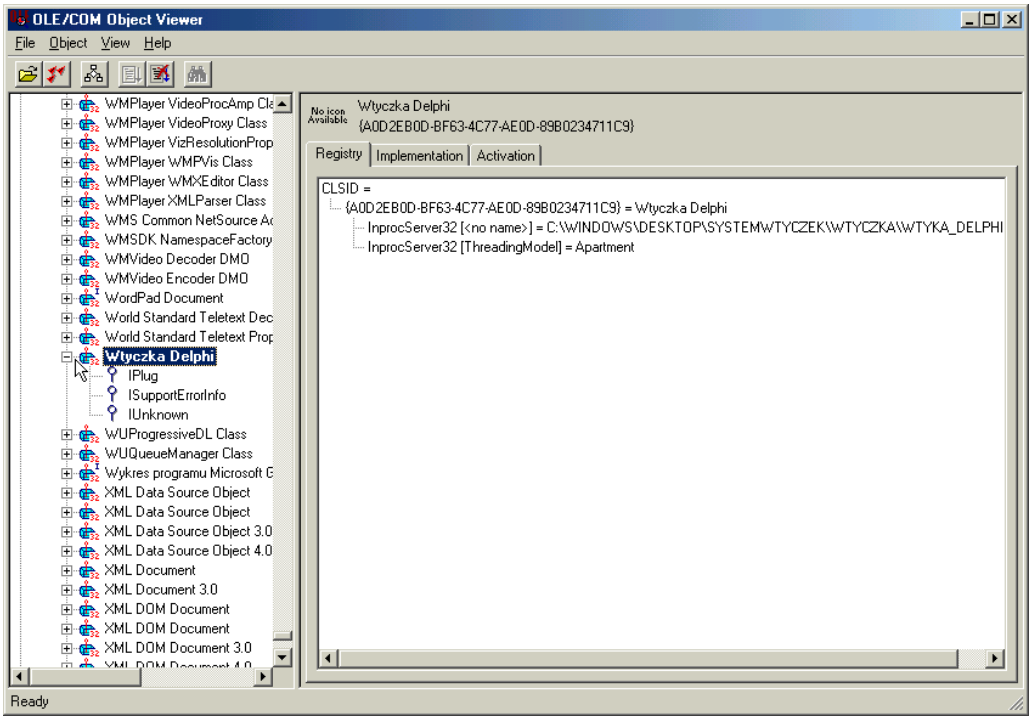
```
type
  TPFSPlugFactory = class(TComObjectFactory)
  public
    procedure UpdateRegistry(Register: Boolean); override;
  end;
```

```
procedure TPFSPlugFactory.UpdateRegistry(Register: Boolean);
var
  ClassID: string;
begin
  if Register then begin
     inherited UpdateRegistry(Register);

      ClassID := GUIDToString(CLSID_DelphiWtyczka);
      with TRegistry.Create do
        try
          RootKey := HKEY_CURRENT_USER;
          OpenKey(szWtyczki+ClassID,True);
          CloseKey;
        finally
          Free;
        end;
  end
  else begin
       with TRegistry.Create do
        try
          RootKey := HKEY_CURRENT_USER;
               ClassID := GUIDToString(CLSID_DelphiWtyczka);
          DeleteKey(szWtyczki+ClassID);
        finally
          Free;
        end;
    inherited UpdateRegistry(Register);
  end;
end;

initialization
CoInitialize(nil);
  TPFSPlugFactory.Create(ComServer, TDelphiWtyczka, CLSID_DelphiWtyczka, '',
      'Wtyczka_Delphi', ciMultiInstance, tmApartment);
finalization
CoUnInitialize;
end.
```

Nasza wtyczka po rejestracji w programie OleView powinna wyglądać jak na rysunku 3 Aby teraz dodać naszą wtyczkę, wystarczy wybrać opcje Zarejestruj z aplikacji rodzica lub też wydać polecenie z linii komend regsvr32 nasza_wtyczka.dll.

6 Możliwości rozwoju

Interfejsy dają niesamowicie łatwą możliwość rozwoju wtyczek. Dodatkowo pozwalają one być kompatybilne wstecz. Jako przykład dołączam drugą wtyczkę, zawierającą dodatkowo implementacje interfejsu IPlugSuper. Interfejs IPlugSuper dziedziczy wszystkie funkcje z Interfejsu IPlug oraz IUnknown dodatkowo dodając funkcje Super.



Rysunek 3: Wtyczka w programie OleView.

Listing 15: Rozszeżona definicja klasy wtyczki SUPER

```
type
  TDelphiSuperWtyczka = class(TComObject,IPlug, IPlugSuper)
  private
   pMalloc: IMalloc;
   Fwind: TSuperForma;
  public
//IPlug
    function Init(ap:IPlugsApplication;Tab:HWND): HResult; stdcall;
    function Mouse(Name:PChar;X,Y:Integer): HResult; stdcall;
    function GetName(var Desc:PChar):HResult; stdcall;
    function FreePlug(): HResult; stdcall;
//IPlugSuper
    function Super(): HResult; stdcall;
 public
    procedure  Initialize; override;
    destructor Destroy; override;
  end;

const

  CLSID_DelphiSuperWtyczka: TGUID = '{0A3963BD-DC55-4BC7-BBC6-C5771891F990}';
```

Banalna implementacja funkcji super.

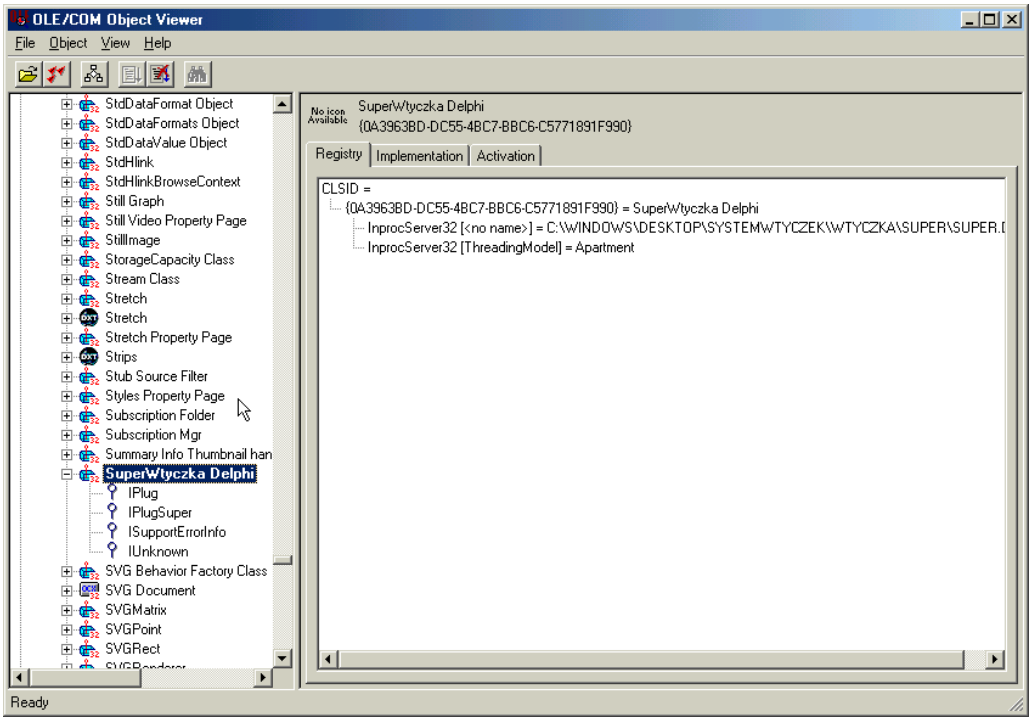
```
  function TDelphiSuperWtyczka.Super(): HResult;
begin
  MessageBox(GetFocus,PChar('Witaj, jestem Super Wtyczka'),'',0);
  Result := S_OK;
end;
```

W drugiej wtyczce przedstawiam sposób w jaki można posługiwać się interfejsem IMalloc do zarządzania pamięcią.

Listing 16: Obsługa pamięci

```
function TDelphiSuperWtyczka.GetName(var Desc:PChar):HResult;
const
  desz  =  'Super wtyczka delphi';
begin
 Desc := pMalloc.Alloc(Length(desz)+1);
 StrCopy(PChar(Desc),desz);
 //rezerwujemy pamięć,
 //aplikacja rodzic jest odpowiedzialna za jej zwolnienie
 Result := S_OK;
end;
```

Nasza SUPER wtyczka po rejestracji w programie OleView powinna wyglądać jak na rysunku 4



Rysunek 4: SUPER Wtyczka w programie OleView.

7 Implemenatacja interfejsu w innych językach

Na razie brak czasu i innego kompilatora. Może są chętni ?

Literatura

[1] K. Brockschmidt. *Inside OLE 2 (2nd ed.)*. Microsoft Press, Redmond, WA, USA, 1995.

[2] X.Pacheco and S.Teixeira. *Delphi 6 Vademecum profesjonalisty Tom II*. Helion (Polish edition), 2002.

[3] Windows 2000 resource kit tool : Ole/com object viewer (oleview.exe). *Microsoft*. http://www.microsoft.com/downloads/details.aspx?FamilyID=5233b70d-d9b2-4cb5-aeb6-45664be858b6&DisplayLang=en.